

Vorlesungsmitschrift

Semantik von Programmiersprachen

gelesen von Prof. Dr. Elfriede Fehr

Tobias Höppner

SoSe 2014

Inhaltsverzeichnis

1	Vorlesung 1 - 17.04.	1
1.1	Was sind Programmiersprachen?	1
1.2	Mehrdeutigkeit in natürlichen Sprachen	1
1.3	Formalisierungsmethoden	1
1.4	Referenzsprache	2
1.4.1	Definition der Syntax	2
2	Vorlesung 2 - 24.04.	4
2.1	Operationelle Semantik am Beispiel der Terme	4
2.1.1	Terme der Sprache WHILE	4
2.1.2	Beispiel: AST	4
2.1.3	Informelle Semantik	4
3	Vorlesung 3 - 08.05.	7
3.1	Operationelle Semantik - allgemeines	7
3.2	Operationelle Semantik von WHILE	7
4	Vorlesung 4 - 15.05.	9
4.1	Reduktionssemantik	9
4.2	Reduktionssemantik der Sprache WHILE	9
4.3	Reduktionssemantik (schematisch)	10
5	Vorlesung 5 - 22.05.	11
5.1	Denotationelle Semantik	11
5.1.1	Allgemeines Prinzip	11
5.1.2	Denotationelle Semantik	11

1 Vorlesung 1 - 17.04.

1.1 Was sind Programmiersprachen?

Definition 1.1 (Programmiersprachen).

Programmiersprachen sind künstliche, formale Ausdruckssprachen zur Kommunikation zwischen Mensch und Maschine.

Memo technischer Begriff -> z.b. ADD reg1 reg2

Beim Studium von Sprachen unterscheidet man 3 Ebenen(Aspekte):

Syntax einschließlich lexikalischer Struktur (Themen des Übersetzerbaus)

- Kern der Syntax ist die grammatikalische Struktur
- formale Definition durch kontextfreie Grammatiken

Semantik (diese Vorlesung)

- Bedeutung
- Interpretation

Natürliche Sprachen (Gegenstand der Geisteswissenschaften) lassen Spielräume zur Interpretation offen. Künstliche Sprachen sollen möglichst formalisierbar sein.

Fokus: Formalisierung

Pragmatik Fragen nach dem Gebrauch und Zweck (Useability).

Warum sagt jemand xyz und ist das leicht verständlich?! - Was will jemand damit bewirken?)

1.2 Mehrdeutigkeit in natürlichen Sprachen

Synonyme *Schloss, Schimmel, ...*

Auflösung durch Kontext (meist leicht und unproblematisch)

Satzebene *Dieses Gelände wird zur Verhütung von Straftaten durch die Polizei Videoüberwacht.*

Auflösung durch Hintergrundwissen möglich. Weiteres Beispiel: *Staatsanwaltschaft ermittelt gegen Betrüger in Clownskostüm.*

1.3 Formalisierungsmethoden

In dieser Vorlesung werden drei Formalisierungsmethoden für die Semantik von Programmiersprachen behandelt.

Motivation

- Sicherheit beim Programmentwurf
- Formale Verifikation von Eigenschaften
- Richtlinie Übersetzerbau
- Automatische Erzeugung von Programm aus Spezifikation

Entwicklung der Formalisierungsansätze

operationale Semantik (*Landin 1964*): Man stützt die Bedeutung auf die Funktionsweise technischer und abstrakte Maschinen ab. Dazu macht man die Maschine so einfach wie möglich und erkläre die Wirkung der Befehle auf die Maschine. Diese Semantik ist ähnlich ähnlich wie die denotationelle Semantik (mathematische Notation), jedoch wirklich näher an der Maschine.

denotationelle Semantik (*McCarthy 1962*): Formales erfassen durch mathematische Notation. Weitgehende Abstraktion vom Zustandsraum mit einer direkten Zuordnung von syntaktischen Komponenten zu mathematischen Objekten (Semantik).

axiomatische Semantik (Hoare 1969): Veränderung/Transformation von Bedingungen/Prädikaten auf dem Zustandsraum (einer abstrakten Maschine). Das geschieht mit mathematischen Formeln. z.B.: Hoareformel: $\{Q\}P\{R\}$

1.4 Referenzsprache

Um alle drei Formalisierungsmethoden zu betrachten nutzen wir die Referenzsprache **WHILE**.

1.4.1 Definition der Syntax

(Wie ist die Sprache grammatikalisch aufgebaut?!)

Elementare Einheiten

```

1 // ganze Zahlen (endlicher Ausschnitt der ganzen Zahlen MIN+1 .. MAX)
2 Z ::= 0 | 1 | ... | MAX | -1 | -2 | ... | MIN
3 // Wahrheitswerte BOOL
4 W ::= TRUE | FALSE
5 // Konstanten KON
6 K ::= Z | W
7 // Bezeichner bzw. Variablen mit Indizes
8 I ::= a | b | ... | z | a1 | a2 | ... | zi
9 // Operatoren
10 OP ::= + | - | * | / | mod
11 // boolesche Operatoren
12 BOP ::= < | > | = | !> | !< | !=

```

Zusätzliche Einheiten (induktiv)

```

1 // Terme TERM
2 T ::= Z | I | T1 OP T2 | READ, für T1, T2 in TERM
3 // boolesche Terme BT
4 B ::= W | T1 BOP T2 | read | not B
5 // Befehle (Zustandstransformation) COM
6 C ::= skip | I := T | C1; C2 | if B then C1 else C2 | while B do C |
   output T | output B

```

Die Indizes sind dazu da das Vorkommen von Symbolen in der Struktur *eindeutig* zu beschreiben.

Warum braucht man für so eine Sprache eine formale Semantik?!

Ich möchte maschinell arbeiten, aber es gibt Unterspezifikationen, unklar ist das Verhalten bei:

- Typkonflikte
- Fehlerbehandlung
- Rekursion

WHILE ist mehrdeutig?

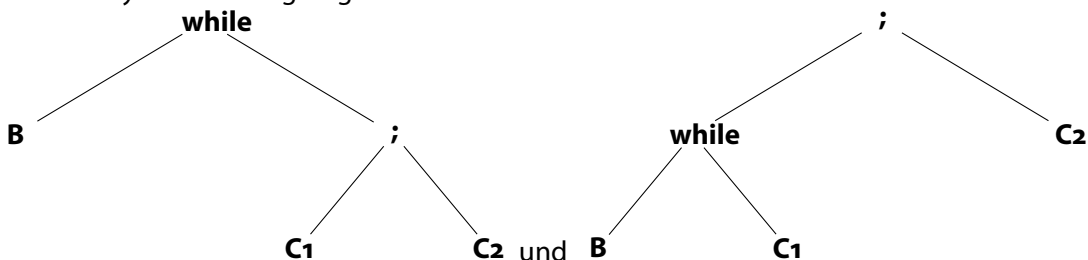
Ja, das zeigt folgendes Beispiel:

```

1 while B do C1; C2

```

wo beide Syntaxbäume gültig sind.



Um dies zu verhindern werden untergeordnete Befehle eingerückt oder geklammert.

```
1 while B do
2   C1;
3   C2
```

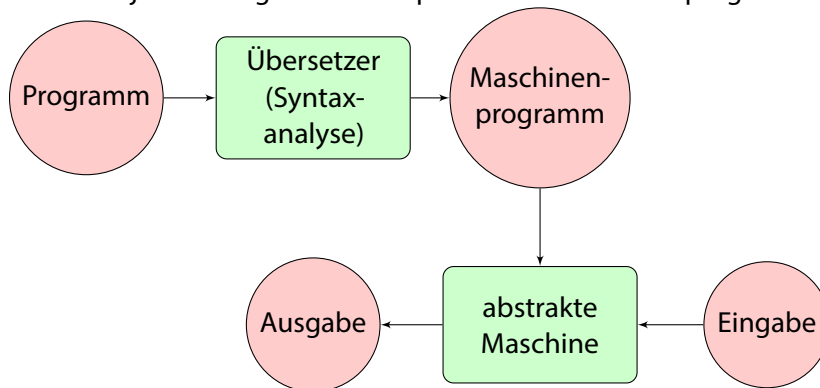
2 Vorlesung 2 - 24.04.

2.1 Operationelle Semantik am Beispiel der Terme

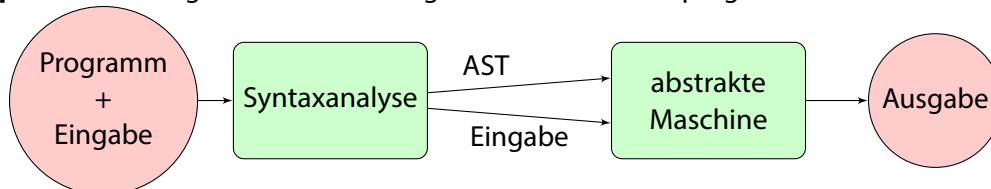
Es ist wichtig die Struktur von einer Sprache zu kennen, erst dann kann man eine korrekte Interpretation anfertigen! *(Inhalt ist nicht im Lehrbuch!)*

Grundsätzlich gibt es zwei Methoden:

Übersetzer Zu jedem Programm ein äquivalentes Maschinenprogramm erstellen.



Interpreter Das Programm wird mit Eingabe zum Maschinenprogramm transferiert.



AST: abstract syntax tree

2.1.1 Terme der Sprache WHILE

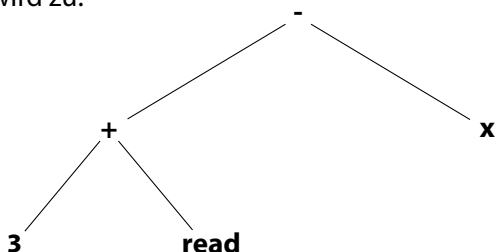
```
1 // Terme TERM
2 T ::= Z | I | T1 OP T2 | READ, für T1, T2 in TERM
```

2.1.2 Beispiel: AST

Der Ausdruck

```
1 3 + read - x
```

wird zu:

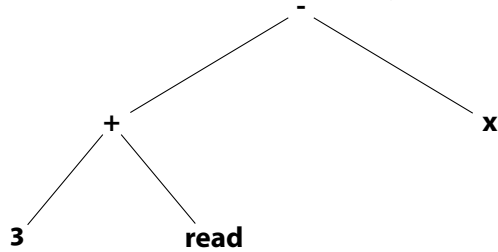


2.1.3 Informelle Semantik

Interpretation nur möglich, wenn Speicher und Eingabe vorgelegt sind.

Annahme: wir bekommen alles als AST und wir bekommen eine Eingabe, die auch von der Maschine unterstützt wird!

Übersetzer Idee: depth-first-left-to-right-postorder Traversierung des AST Unser Beispiel:



wird übersetzt zu:

```

1 PUSH 3
2 READ
3 ADD
4 LOAD x
5 SUB
  
```

Zustandsveränderungen:
aktueller Zustand (siehe Bild Architektur!):

$\langle \epsilon S 8.5 \dots \rangle$	$\xrightarrow{\text{PUSH } 3}$	$\langle 3 S 8.5 \dots \rangle$
	$\xrightarrow{\text{READ}}$	$\langle -8.3 - \epsilon S 5 \dots \rangle$
	$\xrightarrow{\text{ADD}}$	$\langle 3 + (-8) \cdot \epsilon S 5 \dots \rangle$
	$\xrightarrow{\text{LOAD } x}$	$\langle 2 - 5 \cdot \epsilon S \dots 5 \rangle$
	$\xrightarrow{\text{SUB}}$	$\langle -7 \cdot \epsilon S 5 \dots \rangle$

Semantik eines Terms T zu geg. Speicher S und Eingabe E ist die Spitze des Wertekellers (STACK) nach Ausführung von $\text{trans } T$ auf $\langle \epsilon | S | E \rangle$, falls diese Ausführung fehlerfrei läuft, sonst Fehler!

Interpreter (abstrakte Maschine beinhaltet eine Komponente (Kontrollkeller), in der ASTs in einem Keller gespeichert werden können.)

- Kontrollkeller
- Zustand der abstrakten Maschine hat Komponenten
 - * Wertekeller $W (\in Z\text{AHL}^*)$
 - * Speicher $S (S \in [ID \rightarrow Z\text{AHL}])$
 - * Kontrollkeller $K (\in (AST \cup OP)^*)$
 - * Eingabe $E (\in Z\text{AHL}^*)$

Zur Formalisierung der Semantik über die abstrakte Maschine mit dem Zustandsraum Z durch Angabe von:

- (i) einem Anfangszustand $Z_{T,S,E}$ für jeden Term T , Speicher S und Eingabe E .
- (ii) eine Zustandsüberföhrungsfunktion $\Delta : Z \rightarrow Z$ (partiell)
- (iii) Erklärung der Semantik über Iteration von Δ

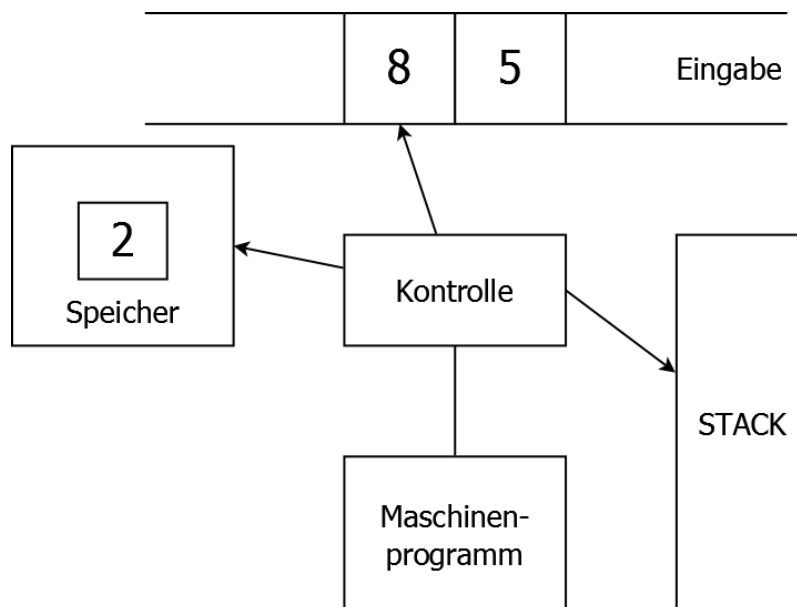
für Terme aus WHILE:

- (i) $Z_{T_0, S_0, E_0} := \langle \epsilon | S_0 | T_0 \cdot \epsilon | E_0 \rangle$
- (ii) Δ per Induktion über die Struktur der Kontrollkellerspitze

$$\begin{aligned}
\Delta \langle W|S|n.K|E \rangle &= \langle n.W|S|K|E \rangle \text{ für alle } n \in ZAHL, W, S, E \text{ wie oben.} \\
\Delta \langle W|S|x.K|E \rangle &= \langle s(x).W|S|K|E \rangle \text{ für alle } x \in ID \\
\Delta \langle W|S|read.K|n.E \rangle &= \langle n.W|S|K|E \rangle \text{ für alle } n \in ZAHL \\
\Delta \langle W|S|T_1OPT_2.K|E \rangle &= \langle W|S|T_1.T_2.OP.K|E \rangle \\
\Delta \langle n_2.n_1.W|S|OP.K|E \rangle &= \langle n_1OPn_2.W|S|K|E \rangle \text{ falls } n_1OPn_2 \text{ definiert ist.}
\end{aligned}$$

- (iii) Die Semantik eines (beliebigen) Terms T im Bezug auf einem Speicher S und eine Eingabe E ist $n \in ZAHL$, wenn $\Delta^k Z_{T,S,E} = \langle n.\epsilon|S|\epsilon|E' \rangle$ für beliebige $E' \in ZAHL^*$, undefiniert sonst!

Architektur der abst. Maschine



Befehlssatz

```

1 // Stack und Speicher Operationen
2 READ      // lese Zahl v. Eingabe, PUSH Zahl, rücke Zeiger um eins weiter
3 LOAD x    // PUSH Inhalt aus Speicher mit symbol. Adr. x
4 PUSH n    // für jede Zahl aus N lege n auf Stack
5 (STORE x) // belegt Speicher mit der symbolischer Adresse x
6 (GOTO n)  // bedingter Sprung
7
8 // arithmetische Operationen
9 ADD
10 MULT
11 SUB

```

3 Vorlesung 3 - 08.05.

(Mitschrift von HvB, da
Autor im Urlaub.
Layout modifiziert von
TH.)

3.1 Operationelle Semantik - allgemeines

Methodik einer Formalisierung (Interpreter) der Semantik einer Programmiersprache \mathcal{P}
Zur operationellen Semantik gehören insbesondere 3 Angaben:

1. Definition des Zustandsraums einer abstrakten Maschine - möglichst einfach: \mathcal{Z}
2. Definition einer (partiellen) Zustandsüberföhrungsfunktion $\Delta : \mathcal{Z} \rightarrow \mathcal{Z}$
3. Definition eines Anfangszustands $Z_{P,E}$ zu jedem Programm P und Eingabe E

Aus 1.-3. ergibt sich die operationelle Semantik \mathcal{O} von \mathcal{P} wie folgt:

$\mathcal{O} : \mathcal{P} \rightarrow [\mathcal{E} \rightarrow \mathcal{A} \cup \{\text{Fehler}\}]$

$$\mathcal{O}(P)(E) = \begin{cases} A, & \text{falls } \exists k \in \mathbb{N} \text{ mit } \Delta^k(Z_{P,E}) = \Delta^{k+1}(Z_{P,E}) \text{ und} \\ & A \text{ die Ausgabekomponente von } Z \text{ ist} \\ \text{Fehler,} & \text{falls es ein } k \in \mathbb{N} \text{ gibt mit } \Delta^k(Z_{P,E}) \text{ nicht definiert} \\ \text{undefiniert,} & \text{sonst.} \end{cases}$$

3.2 Operationelle Semantik von WHILE

1. **Der Zustandsraum \mathcal{Z} ist das kartesische Produkt $\mathcal{W} \times \mathcal{S} \times \mathcal{K} \times \mathcal{E} \times \mathcal{A}$ mit:**

Wertekeller $W \in \mathcal{W}$ ist eine Folge von Konstanten, d.h.

$$\mathcal{W} = \text{KON}^*$$

Speicher $S \in \mathcal{S}$ ist eine Funktion von Bezeichnern nach $\text{ZAHL} \cup \{\text{frei}\}$, d.h.

$$\mathcal{S} = [\text{ID} \rightarrow \text{ZAHL}]$$

Kontrollkeller $K \in \mathcal{K}$ ist ein Folge von ASTs bzw. Kontrollsymbolen, d.h.

$$\mathcal{K} = (\text{TERM} \cup \text{BT} \cup \text{COM} \cup \text{OP} \cup \text{BOP} \cup \{\text{if}, \text{while}, \text{assign}\})^*$$

Ein- und Ausgabe $E \in \mathcal{E}$ bzw. $A \in \mathcal{A}$ ist jeweils eine Folge von Konstanten, d.h.

$$\mathcal{E} = \text{KON}^* \text{ bzw. } \mathcal{A} = \text{KON}^*$$

2. **Induktion über den Aufbau der Kontrollkellerspitze**

a. TERM - $T := Z|I|T_1 \text{ OP } T_2|read$

$$\Delta\langle W|S|n.K|E|A \rangle = \langle n.W|S|K|E|A \rangle \text{ für alle } n \in \text{ZAHL}$$

$$\Delta\langle W|S|x.K|E|A \rangle = \langle S(x).W|S|K|E|A \rangle \text{ für alle } x \in \text{ID mit } S(x) \neq \text{frei}$$

$$\Delta\langle W|S|read.K|n.E|A \rangle = \langle n.W|S|K|E|A \rangle \text{ für alle } n \in \text{ZAHL}$$

$$\Delta\langle W|S|T_1 \text{ OP } T_2.K|E|A \rangle = \langle W|S|T_1.T_2.\text{OP}.K|E|A \rangle$$

$$\Delta\langle n_2.n_1.W|S|\text{OP}.K|E|A \rangle = \langle (n_1 \text{ OP } n_2).W|S|K|E|A \rangle, \text{ falls } n_1 \text{ OP } n_2 \text{ nicht aus dem darstellbaren Zahlenbereich herausführt}$$

b. Boolsche Terme (ähnlich)

c. COM - $C := skip|I := T|C_1; C_2|if B \text{ then } C_1 \text{ else } C_2|while B \text{ do } C|output T|output B$

$$\Delta\langle W|S|skip.K|E|A \rangle = \langle W|S|K|E|A \rangle$$

$$\Delta\langle W|S|I := T.K|E|A \rangle = \langle W|S|T.\text{assign}.I.K|E|A \rangle$$

$$\Delta\langle n.W|S|\text{assign}.I.K|E|A \rangle = \langle W|S|[n/I]|K|E|A \rangle, \text{ wobei } n \in \text{ZAHL und}$$

$$S[n/I](x) = \begin{cases} n, & \text{falls } I = x \\ S(x) & \text{sonst} \end{cases}$$

$$\Delta\langle W|S|C_1; C_2.K|E|A \rangle = \langle W|S|C_1.C_2.K|E|A \rangle \Delta\langle W|S|if B \text{ then } C_1 \text{ else } C_2.K|E|A \rangle =$$

$$\langle W|S|B.\text{if}.C_1.C_2.K|E|A \rangle$$

$$\begin{aligned}
\Delta\langle \underline{true}.W|S|\underline{if}.C_1.C_2.K|E|A\rangle &= \langle W|S|C_1.K|E|A\rangle \\
\Delta\langle \underline{false}.W|S|\underline{if}.C_1.C_2.K|E|A\rangle &= \langle W|S|C_2.K|E|A\rangle \\
\Delta\langle W|S|\underline{while}\ \underline{B}\ \underline{do}\ C.K|E|A\rangle &= \langle W|S|B.\underline{while}.B.C.K|E|A\rangle \\
\Delta\langle \underline{true}.W|S|\underline{while}.B.C.K|E|A\rangle &= \langle W|S|C.B.\underline{while}.B.C.K|E|A\rangle \\
\Delta\langle \underline{false}.W|S|\underline{while}.B.C.K|E|A\rangle &= \langle W|S|K|E|A\rangle
\end{aligned}$$

$$\begin{aligned}
\Delta\langle W|S|\underline{output}\ T.K|E|A\rangle &= \langle W|S|T.\underline{output}.K|E|A\rangle \\
\Delta\langle n.W|S|\underline{output}.K|E|A\rangle &= \langle W|S|K|E|n.A\rangle
\end{aligned}$$

4 Vorlesung 4 - 15.05.

4.1 Reduktionssemantik

Ausprägungen der operationellen Semantik zu einfacheren Argumentation (Beweisführung) über Programmeigenschaften.

Idee: Reduktion von Ausdrücken, Termen, Programmen und Anweisungen (usw.) auf einfachere aber semantisch äquivalenten Termen (usw.).

Beispiel: Einfache arithmetische Ausdrücke über den natürlichen Zahlen und $+$, $*$.

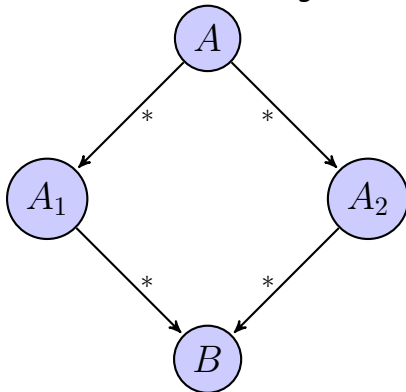
Einzelschrittreduktion: $(4 + 2) * (7 - 5) \Rightarrow 6 * (7 - 5) \Rightarrow 6 * 2 \Rightarrow 12$

allgemeine Form: $A_1 \underline{OP} A_2 \Rightarrow A'_1 \underline{OP} A'_2$ falls $A_1 \Rightarrow A'_1$ und $A_2 \Rightarrow A'_2$

Axiom: $n_1 \underline{OP} n_2 \Rightarrow n_1 \underline{OP} n_2$

Alternative Schreibweise(Winskel): $\frac{A_1 \Rightarrow A'_1, A_2 \Rightarrow A'_2}{A_1 \underline{OP} A_2 \Rightarrow A'_1 \underline{OP} A'_2}$

Konfluenz: (Church-Rosser Eigenschaft, Diamant)



Church-Rosser gilt nicht bei Nebenwirkungen!

anderes Beispiel: λ -Kalkül. $(\lambda x.M)A \xrightarrow{\beta} M$ falls...

4.2 Reduktionssemantik der Sprache WHILE

Zustandsraum: $\mathcal{Z} = \rho \times \mathcal{E} \times \mathcal{A}$ Speicher, Ein- und Ausgabe. $\rho = [ID \rightarrow \text{ZAHL} \cup \{\text{frei}\}]$, $\mathcal{E} = \text{KON}^*$, $\mathcal{A} = \text{KON}^*$

\Rightarrow Reduktionsrelation über $(\text{TERM} \cup \text{BT} \cup \text{COM}) \times \mathcal{Z}$

Induktiv über den Aufbau der Syntax:

1. **Terme:** Keine Reduktionsregel für (n, z) , d.h. Normalform für $n \in \text{ZAHL}$

a $(x, (s, e, a)) \Rightarrow (s(x), (s, e, a))$, falls $s(x) \neq \text{frei}$ für $x \in ID$, $(s, e, a) \in \mathcal{Z}$

b $\text{read} \Rightarrow (n, (s, e, a))$, falls $n \in \text{ZAHL}$.

c $(T_1 \underline{OP} T_2, z) \Rightarrow (n \underline{OP} T_2, z')$, falls $(T_1, z) \xRightarrow{*} (n, z')$

d $(n \underline{OP} T_1, z) \Rightarrow (n \underline{OP} m, z')$, falls $(T, z) \xRightarrow{*} (m, z')$

e $(n \underline{OP} n, z) \Rightarrow (n \underline{OP} m, z')$, falls $n \underline{OP} m \in \text{ZAHL}$.

2. **BT** analog.

3. **COM:** keine Reduktionsregel für skip (Normalform)

a $(I := T, (s, e, a)) \Rightarrow (\text{skip}, (s[n/I], e', a))$, falls $(T, (s, e, a)) \xRightarrow{*} (n, (s, e', a))$

b $\text{output } T, (s, e, a) \Rightarrow (\text{skip}, (s, e', a.n))$, falls $(T, (s, e, a)) \xRightarrow{*} (n, (s, e', a))$

c $\text{output } B, (s, e, a) \Rightarrow (\text{skip}, (s, e', a.b))$, falls $(B, (s, e, a)) \xRightarrow{*} (b, (s, e', a))$

d $(C_1; C_2, z) \Rightarrow (C_2, z)$, falls $(C_1, z) \xRightarrow{*} (\text{skip}, z')$

e $(\text{if } B \text{ then } C_1 \text{ else } C_2, z) \Rightarrow (C_1, z')$, falls $(B, z) \xRightarrow{*} (\text{true}, z')$

f $(\text{if } B \text{ then } C_1 \text{ else } C_2, z) \Rightarrow (C_2, z')$, falls $(B, z) \xRightarrow{*} (\text{false}, z')$

g $(\text{while } B \text{ do } C, z) \Rightarrow (C; \text{while } B \text{ do } C, z')$, falls $(B, z) \xRightarrow{*} (\text{true}, z')$

h $(\underline{while} \ B \ \underline{do} \ C, z) \Rightarrow (\underline{skip}, z'), \text{ falls } (B, z) \Rightarrow^* (\underline{false}, z')$

4.3 Reduktionssemantik (schematisch)

$$\underline{eval}(P)(E) = \begin{cases} A, & \text{falls } (P, (S_0, E, \mathcal{E})) \Rightarrow^* (\underline{skip}, (S, E', A)) \text{ mit bel. } S \in \rho \text{ und } E', A \in \text{KON}^* \\ \underline{\text{Fehler}}, & \text{falls } (P, (S_0, E, \mathcal{E})) \Rightarrow^* (C, (S, E', A)) \text{ mit bel. } C \in \text{COM} \text{ und } E', A \in \text{KON}^* \\ & \text{und } C \neq \underline{skip} \text{ und } (C, (S, E', A)) \text{ lässt sich nicht mehr mit } \Rightarrow \text{ reduzieren} \\ \text{undefiniert,} & \text{sonst.} \end{cases}$$

Satz: $\mathcal{O} = \underline{eval}$ (extensional)

Beweis über strukturelle Induktion.

5 Vorlesung 5 - 22.05.

5.1 Denotationelle Semantik

Grundprinzip:

Direkte (injektive) Zuordnung von syntaktischen Strukturen zu deren Semantik (als mathematisches Objekt (Konstrukt)).

5.1.1 Allgemeines Prinzip

1. Ordne jeder syntaktischen Kategorie K der Sprache einen semantischen Bereich B_K zu.
2. Schreibe (definiere) zu jeder syntaktischen Kategorie eine Semantikfunktion \mathcal{K}

$$\mathcal{K} : K \rightarrow B_K$$

Notation:

- Das Argument einer Semantikfunktion \mathcal{K} schreiben wir in \llbracket und \rrbracket
- Funktionen z.B. $f : A \rightarrow (B \rightarrow C)$ werden $f \ a \ b = a \dots b \dots$

(Klammern sparen!)

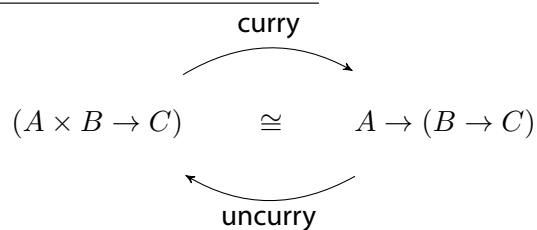
5.1.2 Denotationelle Semantik

- **Syntaktische Kategorien** von WHILE: TERM, BT, COM, PROG
- Hilfsbereiche:
ZUSTAND = SPEICHER \times EINGABE \times AUSGABE mit SPEICHER = $ID \rightarrow ZAHL \cup \{\underline{frei}\}$,
EINGABE = KON^* und AUSGABE = KON^*

(Abh. vom Speicher + Nebenwirkung)

1. $\mathcal{T} : TERM \rightarrow \underline{\underline{B_T}}$
 $B_T := ZAHL ???$

Exkurs: Curry-Isomorphie



curry: $f \ a \ b = f(a, b)$

uncurry: $g \ (a, b) = g \ a \ b$

$B_T := \text{SPEICHER} \rightarrow \text{EINGABE} \rightarrow ((ZAHL \times \text{Eingabe}) \cup \{\underline{Fehler}\})$

$\mathcal{B} : BT \rightarrow \text{SPEICHER} \rightarrow \text{EINGABE} \rightarrow ((\text{BOOL} \times \text{Eingabe}) \cup \{\underline{Fehler}\})$

$\mathcal{C} : COM \rightarrow \underbrace{ZUSTAND \rightarrow ZUSTAND \cup \{\underline{Fehler}\}}_{B_C}$

2. Induktive Definition über dem Aufbau

$$\begin{aligned}
\mathcal{T}[\![n]\!] s e &= (n, e) \quad \text{für alle } n \in \text{ZAHL} \\
\mathcal{T}[\![x]\!] s e &= \begin{cases} (n, e), & \text{falls } s x = n \\ \text{Fehler}, & \text{falls } s x = \text{frei für alle } x \in \text{ID} \end{cases} \\
\mathcal{T}[\![\text{read}]\!] s e &= \begin{cases} (n, e'), & \text{falls } e = n.e' \\ \text{Fehler}, & \text{falls } e = \mathcal{E} \text{ oder } e = b.e' \text{ mit } b \in \text{BOOL} \end{cases} \\
\mathcal{T}[\![T_1 \text{ OP } T_2]\!] s e &= \begin{cases} (n, e''), & \text{falls } \mathcal{T}[\![T_1]\!] s e = (n_1, e') \text{ und} \\ & \mathcal{T}[\![T_2]\!] s e' = (n_2, e'') \text{ und} \\ & n_1 \text{ OP } n_2 = n \in \text{ZAHL} \\ \text{Fehler}, & \text{sonst} \end{cases}
\end{aligned}$$

$$\begin{aligned}
\mathcal{B}[\![w]\!] s e &= (w, e) \quad \text{für alle } w \in \text{BOOL} \\
\mathcal{B}[\![\text{not } B]\!] s e &= \begin{cases} (b, e'), & \text{falls } \mathcal{B}[\![B]\!] s e = (w, e') \text{ und } b = \neg w \\ \text{Fehler}, & \text{sonst} \end{cases} \\
\mathcal{B}[\![T_1 \text{ BOP } T_2]\!] s e &= \begin{cases} (b, e''), & \text{falls } \mathcal{T}[\![T_1]\!] s e = (n_1, e') \text{ und} \\ & \mathcal{T}[\![T_2]\!] s e' = (n_2, e'') \text{ und} \\ & n_1 \text{ BOP } n_2 = b \in \text{BOOL} \\ \text{Fehler}, & \text{sonst} \end{cases} \\
\mathcal{B}[\![\text{read}]\!] s e &= \begin{cases} (b, e'), & \text{falls } e = b.e' \text{ mit } b \in \text{BOOL} \\ \text{Fehler}, & \text{falls } e = \mathcal{E} \text{ oder } e = n.e' \text{ mit } n \in \text{ZAHL} \end{cases}
\end{aligned}$$

$$\begin{aligned}
\mathcal{C}[\![\text{skip}]\!] z &= z & \text{alt. Schreibweise: } \mathcal{C}[\![\text{skip}]\!] z &= \text{id} \\
\mathcal{C}[\![I := T]\!] (s, e, a) &= \begin{cases} (S[n/I], e', a), & \text{falls } \mathcal{T}[\![T]\!] s e = (n, e') \\ \text{Fehler}, & \text{sonst} \end{cases} & z &= (s, e, a) \\
\mathcal{C}[\![\text{output } T]\!] (s, e, a) &= \begin{cases} (s, e', a.n), & \text{falls } \mathcal{T}[\![T]\!] s e = (n, e') \\ \text{Fehler}, & \text{sonst} \end{cases} \\
\mathcal{C}[\![\text{output } B]\!] (s, e, a) &= \begin{cases} (s, e', a.b), & \text{falls } \mathcal{B}[\![B]\!] s e = (b, e') \\ \text{Fehler}, & \text{sonst} \end{cases} \\
\mathcal{C}[\![C_1; C_2]\!] z &= \begin{cases} \mathcal{C}[\![C_2]\!] z', \mathcal{C}[\![C_1]\!] z = z' & \text{mit } z' \in \text{ZUSTAND} \\ \text{Fehler}, & \text{sonst} \end{cases} \\
\mathcal{C}[\![\text{if } B \text{ then } C_1 \text{ else } C_2]\!] (s, e, a) &= \begin{cases} \mathcal{C}[\![C_1]\!] (s, e', a), & \text{falls } \mathcal{B}[\![B]\!] s e = (\text{true}, e') \\ \mathcal{C}[\![C_2]\!] (s, e', a), & \text{falls } \mathcal{B}[\![B]\!] s e = (\text{false}, e') \\ \text{Fehler}, & \text{sonst} \end{cases} \\
\mathcal{C}[\![\text{while } B \text{ do } C]\!] (s, e, a) &= \begin{cases} \mathcal{C}[\![C; \text{while } B \text{ do } C]\!] (s, e', a), & \text{falls } \mathcal{B}[\![B]\!] s e = (\text{true}, e') \\ (s, e', a), & \text{falls } \mathcal{B}[\![B]\!] s e = (\text{false}, e') \\ \text{Fehler}, & \text{sonst} \end{cases} \\
\mathcal{P}[\![P]\!] e &= \begin{cases} a, & \text{falls } \mathcal{C}[\![C]\!] (s_0, e, \epsilon) = (s, e', a) \\ \text{Fehler}, & \text{falls } \mathcal{C}[\![C]\!] (s_0, e, \epsilon) = \text{Fehler} \\ \text{undefiniert}, & \text{sonst} \end{cases}
\end{aligned}$$